OULUN SEUDUN
AMMATTIKORKEAKOULU

Jiri Lapinoja

# AUTOMAATTISEN ASIAKASTIEDON KERÄYKSEN TOTEUTUS MYPOSE-SOVELLUKSEEN

# AUTOMAATTISEN ASIAKASTIEDON KERÄYKSEN TOTEUTUS MYPOSE-SOVELLUKSEEN

Jiri Lapinoja
Opinnäytetyö
Kevät 2013
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

# TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehitys

---

Tekijä: Jiri Lapinoja
Opinnäytetyön nimi: Automaattisen asiakastiedon keräyksen toteutus Mypose sovellukseen
Työn ohjaaja: Pertti Heikkilä
Työn valmistumislukukausi ja -vuosi: Kevät 2013
Sivumäärä: 59 + 1 liite

---

Tässä opinnäytteessä tuotettiin itsenäisesti toimiva ohjelmistokokonaisuus, joka kerää tietoa Mypose-laitteen käytöstä ja käyttäjistä. Tallennusta varten suunniteltiin ja toteutettiin tietokanta. Työn tilasi Mypose Oy.

Mypose-sovellus on kehitetty Qt-kielellä, johon luotiin uusi luokka hoitamaan statistiikkaa. Työn tavoite oli tuottaa palvelu, jolla Myposen asiakasyritykset voivat saada tietoa käyttäjistään. Ohjelma kerää tietoa käyttäjien iästä, paikasta ja sukupuolesta sekä laitteen käytöstä kuvien otto- ja lähetysmäärinä.

Tiedonkeräämisessä käytetään Facebook-sovellusta, jonka avulla käyttäjä voi lähettää Myposesta kuvan omalle Facebook-seinälleen.

Työ suoritettiin Qt-, PHP- ja MyQSL-kielillä. Ohjelmointiin käytettiin Qt Creatoria ja Gedit-tekstinmuokkainta.

Toteutuksessa vastaan tuli eettisiä kysymyksiä tietojen keruusta ja oli aiheellista tutustua tietosuojakäytöntöihin ja henkilötietolakiin. Henkilöiden yksityisyyttä vaalittiin, eikä käyttäjien nimiä tai muita henkilötietoja otettu työssä talteen.

Työn tavoitteet toteutuivat odotetusti. Tiedonkeruu on automaattista ja kunnioittaa käyttäjien yksityisyyttä. Jatkokehityksessä olisi mahdollista tuottaa kerätyistä tiedoista graafisia tuloksia asiakasyrityksille esiteltäviksi.

---

Asiasanat:

käyttäjätieto, statistiikka, tietokannat, Facebook, Mypose

# ABSTRACT

Oulu University of Applied Sciences
Degree programme in information technology, Software development

---

Author: Jiri Lapinoja
Title of thesis: Creating a solution for automated user data collecting for Mypose application
Supervisor: Pertti Heikkilä
Term and year when the thesis was submitted: Spring 2013
Pages: 59 + 1 appendices

---

In this thesis a self functional software was developed which collects information about the users of Mypose device. A database was designed and created for storing the data. The thesis was ordered by Mypose Ltd.

Mypose application has been created using Qt programming language in which a new class was developed to manage statistics. The goal for this thesis was to produce a service for Mypose's clients to get information about devices users. The collected information contains age, location and sex of users as well as amounts of pictures taken and send from Mypose device.

Facebook application is used for collecting user information. The application is used to send pictures from Mypose device to users personal Facebook wall.

The software development was created using Qt (by QtCreator and Gedit), PHP and MySQL languages.

During the process some ethical questions came up about collecting user data and it was necessary to familiarize oneself with privacy policies as well as Finnish laws about collecting personal data. Privacy was kept in mind and personal data was not collected.

The goals were achieved. data collection is made automatic while maintaining respectful against users privacy. In further development the collected data could be transferred into bars and graphs for easier readability.

---

Keywords:
User data, statistics, database, Facebook, Mypose

# ALKULAUSE

Työ on nyt tehty kunnialla loppuun. Kiitän Mypose Oy:tä, toimitusjohtajaa Lassi Anttosta sekä työn ohjaajaa Pertti Heikkilää. Lisäksi motivoinnista ja tuesta haluan kiittää työtoveriani Juho Juntusta sekä avopuolisoani Satu Volotista.

Oulussa 12.5.2013

Jiri Lapinoja

# SISÄLLYS

## SANASTO

| | |
|---|---|
| **ER-kaavio** | Entity-relationship diagram. Tietokantojen piirtämiseen käytetty kaaviomalli. |
| **FTP** | File Transfer Protocol. Protokolla tiedostojen siirtämiseen verkon välityksellä. |
| **Qt** | Alustasta riippumaton C++ kieleen perustuva ohjelmointikieli. |
| **SDK** | Software Development Kit. Ohjelmankehitysalusta. |
| **Smtp** | Simple Mail Transfer Protocol. Sähköpostinvälitysprotokolla. |
| **Staattinen metodi** | Metodi, jota on mahdollista käyttää ilman luokan oliota. |

# 1 JOHDANTO

Käyttäjätiedon keräämisellä tarkoitetaan yrityksen tuottaman palvelun tiedonkeruuta käyttäjistä yleensä kaupalliseen tarkoitukseen. Kiinnostavaa informaatiota voi olla lähes mikä tahansa tieto käyttäjästä. Yleisimmin kerättävää tietoa voidaan käyttää segmentoimaan ihmisiä. Näitä tietoja voivat olla ikä, sukupuoli, sijainti ja mahdollisesti yhteystiedot. Tähän informaatioon nojaa yritysten mainonta.

Käyttäjätietoa keräävien sovellusten kirjo on laaja. Esimerkiksi tietotekniikan jättiläiset Google ja Facebook saavat käyttäjistään äärimmäisen tarkkaa tietoa. Facebook tarjoaa sovelluskehittäjilleen työkaluja tiedon saamiseen sovelluksen käyttäjästä, jos itse Facebook on saanut käyttäjältä kyseisen tiedon. Käyttäjä itse antaa sovellukselle luvan käyttää näitä tietoja sallimalla sovelluksen käyttöoikeussopimuksen.

Mypose on sovellus interaktiiviselle infonäytölle, jolla on mahdollista ottaa itsestään kuvia ja videokuvaa ja lähettää niitä Facebookiin tai omaan sähköpostiin. Mypose-kokonaisuus pitää sisällään suuren kosketusnäytön, web-kameran, jalustan, tietokoneen ja Mypose-sovelluksen.

Tämän opinnäytetyön tavoite on tutustua käyttäjätiedon keräämisen metodeihin ja toteuttaa Mypose-sovellukselle käyttäjästatistiikkaa keräävä kokonaisuus, joka toimii automaattisesti sovellusta käytettäessä. Saadut tiedot tallennetaan MySQL-tietokantaan, josta ne ovat luettavissa ja järjesteltävissä helposti. Yksi tärkeä kriteeri työlle on pystyä yksilöimään jokainen Mypose-kokonaisuus.

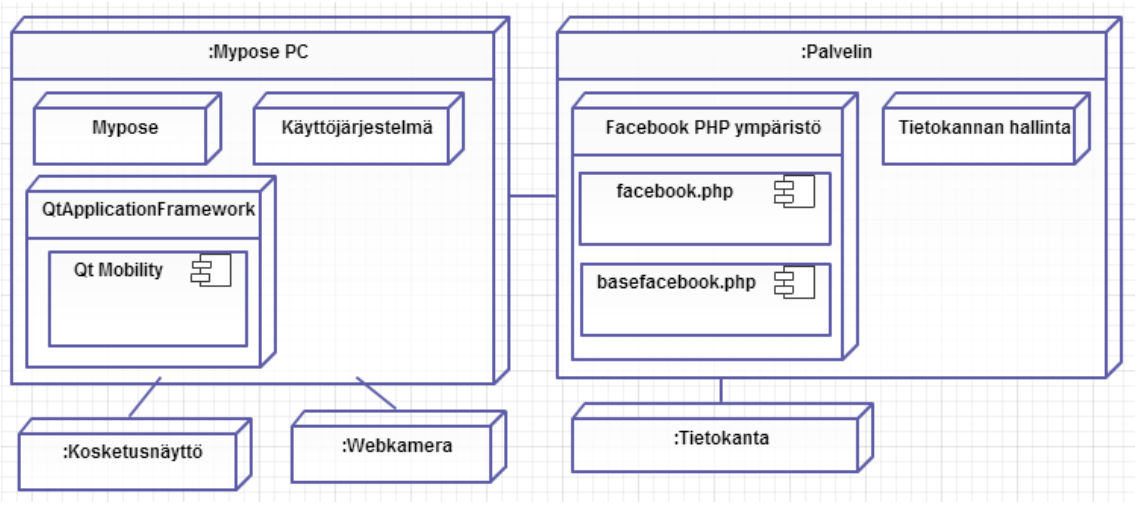
# 2 MYPOSE JA SEN TOIMINTA

Mypose on suuri kosketusnäyttö, joka näyttää reaaliaikaista videokuvaa edestään peilin tavoin. Näyttö on asetettu pystysuuntaan jalustalle ja sen laitaan on kiinnitetty web-kamera. Näytön toimintaa ohjaa tietokone, jossa pyörii Qt:lla kirjoitettu sovellus. Käyttöliittymässä on tavoiteltu keveyttä. Siihen on päästy käyttämällä osittain läpinäkyviä neutraaleja painonappeja ja koko ruudun kokoista videokuvaa. Seuraavassa kuvassa näytetään Mypose-laite vaatekaupassa.

*KUVA 1. Mypose-laite vaatekaupassa*

Mypose-laitteella käyttäjät voivat verrata vaatteita vaatekaupassa ottamalla itsestään kuvia, joissa heillä on päällä vaatekaupan vaatteita. Vertailu onnistuu kuvakirjastoa selaamalla. Kuvia on mahdollista lähettää itselle tai tutulle sähköpostin välityksellä. Jos käyttäjä haluaa, voi hän myös tallentaa kuvan omaan Facebook-profiiliinsa. Tallentaminen onnistuu suoraan ohjelmasta kirjautumalla omilla tunnuksilla Facebookiin. Kaikkiin lähetettyihin kuviin tallennetaan vesileiman kaltaisesti liikkeen logo. Sosiaalisessa mediassa näkyminen on Myposen tarjoama hyöty liikkeille. Vaatteen sovitusta helpottaa muutaman sekunnin

viivästetty livekuva, jolloin on esimerkiksi mahdollista pyörähtää laitteen edessä ja nähdä, miltä vaate näyttää takaa, mikä on hankala katsoa normaalilla peilillä. Myposen käyttöönottokaaviosta saa kuvan laitteen kokonaisuudesta.




*KUVA 2. Myposen käyttöönottokaavio*

Kuten käyttöönottokaaviosta nähdään, Mypose keskustelee palvelimen kanssa. Palvelin huolehtii Facebook-yhteydestä sekä ylläpitää tietokantaa laitteiden käytöstä. Osa tästä opinnäytteestä toteuttaa tuon tietokannan. Facebookia varten Mypose-laite siirtää kuvia FTP:n (File Transfer Protocol) välityksellä palvelimelle, josta ne ladataan käyttäjän Facebook-profiiliin. Kun transaktio on onnistunut, poistetaan kuva palvelimelta. Jos kuvan lähetys keskeytetään, poistetaan kuva myös palvelimelta. Tällä tavoin varmistetaan, ettei käyttäjän kuvia jää paikkoihin, joihin käyttäjä ei itse niitä haluaisi tallentaa.

# 3 KÄYTTÄJÄTIETO

## 3.1 Merkitys

Käyttäjätiedolle tunnetaan useita määritelmiä, joista useimmat jakavat Hyysalon (1, s. 12–13) ajatuksen siitä, että käyttäjätieto on käyttäjiä ja käyttöä koskevaa luotettavaa tietoa. Käyttäjätietoa tarvitaan erityisesti tuotekehityksessä. Viime vuosina asiakkaat ovat kehittyneet entistä tuotetietoisimmiksi ja kiinnostuneemmiksi ostamistaan palveluista. Joissakin tilanteissa asiakas ei ole tuotteen loppukäyttäjä, jolloin käyttäjätieto voi olla arvokasta sekä tuotteen kehittäjälle että asiakkaalle. Esimerkiksi kolikkopelit eivät ole huoltoaseman omistamia vaan palvelu huoltoasemalle, joka maksaa kolikkopelien tuottajalle vuokraa laitteista. Tässä tapauksessa käyttäjätieto on hyvin tärkeää huoltoaseman omistajalle. Häntä kiinnostavaa tietoa ovat pelikoneen tuomat voitot asiakkaiden määrissä ja se, kuinka kauan he viipyvät kaupassa. Tärkein tutkittava tieto on, kuinka paljon asiakaskunta ostaa silloin, kun huoltoasemalla on käytettävissä kolikkopeli. Tästä voidaan myös laskea vuokrasuhteen kannattavuus ja tarvittaessa myös vuokran määrä. Tällainen tieto on suoraan arvokasta yritykselle ja siitä voidaan veloittaa siinä missä kolikkopeleistäkin.

Käyttäjä ei aina ole tietoinen hänestä kerätyn tiedon määrästä tai laadusta. Pääasiallisesti voidaan ajatella, että käyttäjätietoa kerää jokainen palvelu. Poikkeuksena voidaan laskea palvelut ja laitteet, joille ei enää tapahdu mitään kehitystä tai niitä ylläpitävä yritys ei enää ole toiminnallinen.

## 3.2 Menetelmiä käyttäjätiedon keräämiseen

Käyttäjätietoa tutkitaan monin erilaisin tavoin käyttötarkoituksen mukaan. Tuotekehitykseen soveltuvia tapoja ovat aktiiviset tavat, kuten käytettävyystestit, kyselyt, haastattelut, teknisen tuen ylläpito, kilpailijoiden analysointi ja fokusryhmien kokoaminen. Passiivisia metodeja voidaan käyttää tuotekehityksessä, mutta niitä voidaan soveltaa myös kaupallisissa tarkoituksissa, kuten tässä nimenomaisessa tapauksessa. Passiivisia metodeja voivat olla lokien tutkiminen tai asiakasrekisterin kerääminen. Tähän työhön valitsin passiivisen lähestymisen, koska työn tarkoitus on tuottaa sisältöä tilaajayhtiölle.

Tässä työssä käytetään pääsääntöisesti lokityylistä informaatiota. Alun perin lokit tai lokikirjat tulevat portugalilaisten merenkävijöiden tavasta pitää salaista kirjaa merenkulkureiteistä (2). Sittemmin termi on otettu tietotekniikassa käyttöön kuvaamaan ohjelmassa tapahtuvia tärkeitä muutoksia, käyttäjien tekemiä pyyntöjä esimerkiksi verkkopalvelimelta, ohjelman kaatumisia tai mitä tahansa, mistä voidaan saada yksiselitteinen merkintä lyhyessä muodossa. Lokien vahvuus on siinä että hyvin toteutettuna ne voivat antaa tarkan kuvan tuotteen käytöstä (3, s.395). Useimmin ongelmia ei tulekaan toiminnan tarkkuuden kuvaamisessa, vaan tärkeän informaation kaivamisessa informaatiotulvasta. Tässä työssä pyritään vastaamaan kysymyksiin: Kuka lähetti kuvan? Kuinka monelle hän lähetti sen? Kuinka monta kuvaa yhdellä laitteella otetaan? Mitkä ovat kiireisimpiä aikoja käyttää laitetta yhdessä paikassa? Minkä ikäiset tykkäävät käyttää laitetta? Miehet vai naiset ottavat kuvia? Teknisesti olisi suunnaton työ tehdä erilaisia tekniikoita tunnistamaan miehet naisista tai ikä kuvasta. Siksi tarttumapintana käytetään Facebookin tarjoamia funktioita. Teoriassa voidaan lokitiedoista päätellä kuka teki mitä ja milloin (3, s.404).

# 4 FACEBOOK-SOVELLUSTEN YKSITYISYYDENSUOJA

Facebook tarjoaa kehittäjilleen useita SDK:ita (Software Develpment Kit), joita kehittävät voivat vapaasti käyttää. Facebook developers -sivu antaa ohjeita ja tutoriaaleja JavaScript, PHP, iOS3.0 ja Android SDK:ille. Tämä työ käyttää Facebookin PHP-tukea. Facebook tuo myös haasteita, koska itse lähdekoodiin ei ole mahdollista päästä käsiksi. On käytettävä Facebookin rajapintaa. Rajapinta pitää sisällään metodeja, joilla saadaan haettua käyttäjästä hänen syöttämiään tietoja sovelluksen tarpeisiin.

## 4.1 Käyttöehdot

Kerätessä käyttäjätietoa Facebook-sovelluksen avulla, on muistettava käyttäjien yksityisyydensuoja. Facebookilla on hyvin laaditut yksityisyydensuojadokumentit, jotka kertovat käyttäjälle, mitä tietoja Facebook jakaa sovelluksille sekä mikä on kaikkien muidenkin nähtävissä. Ilman erillistä ilmoitusta käyttäjälle, sovellukset saavat käyttäjästä käyttäjätunnuksen, kavereiden käyttäjätunnukset ja julkiset tiedot. Sovelluksen on mahdollista saada myös näiden nimettyjen tietojen ulkopuolelle jääviä tietoja eritysluvalla, jota pyydetään, kun sovellusta käytetään ensimmäistä kertaa. Jos käyttäjä ei hyväksy tietojen luovuttamista, ei hän voi käyttää kyseistä sovellusta osittain tai kokonaan. Käyttäjä voi lisäksi hallita asennetulle ohjelmalle annettavia tietoja tai poistaa koko ohjelman sovellusasetuksissa. Jos sovelluksen poistaa tai siltä vie kaikki oikeudet, ei sovellus enää saa tietoja, joita se on käyttöönotettaessa saanut. (4.)

## 4.2 Julkiset ja yksityiset tiedot

Julkisia tietoja ovat käyttäjän itse julkaisemat tiedot, kuten etu- ja sukunimi, sähköpostiosoite, syntymäaika ja sukupuoli. Lisäksi julkisiksi tiedoiksi Facebook mainitsee kaiken tiedon, jonka käyttäjä itse jakaa. Kaikkea tietoa, jota Facebook käyttäjästä kerää, ei jaeta sovelluksille. Sovelluksen joutuvat esimerkiksi pyytämään syntymäaikaa, huolimatta siitä että Facebook luokitteleekin sen julkiseksi tiedoksi. Yksityisiä tietoja ovat kaikki Facebookin keräämät tiedot, jotka eivät ole julkisia. (4.)

### 4.3 Tietojen käyttöön liittyvät rajoitukset ja määräykset

Facebookista on löydettävissä alustaa koskevat linjaukset, joita tulee tutkia ennen sovelluksen julkaisemista. Linjaukset on kirjoitettu englanniksi ja ne ovat myös saatavilla muutamilla muilla kielillä. Suomeksi ei dokumenttia ole käännetty. Käännöksissä voi olla tulkinnallisia eroavaisuuksia, joista koituvissa ristiriidoissa englanninkielinen dokumentti on määräävä. (4.)

Pääpiirteittäin linjauksessa korostetaan sovelluksen olevan sen käyttäjiä varten, mikä tarkoittaa, että käyttäjä hallitsee sovellusta ja sovelluksella on mielekäs tarkoitus. Lisäksi sovelluksen kehittäjiä kehotetaan luotettavuuteen. Käyttäjien yksityisyyttä on kunnioitettava. Käyttäjien harhaanjohtaminen tai pettäminen on kiellettyä. Kaiken kaikkiaan sovelluksen tulee palvella käyttäjää, käyttäjän ei sovellusta. (4.)

# 5 TIETOKANNAT

Tietokanta on kokoelma tietoja, jotka liittyvät toisiinsa. Tietokantojen koot vaihtelevat ihan pienistä, muutaman rivin tietokannoista, useiden miljoonien tietueiden tietokantoihin. Tietokanta ei tarkoita ainoastaan sähköisesti tietokantapalvelimelle tallennettua tietoa; jo yksinkertainen puhelinmuistio on tietokanta. Tietotekniikan yhteydessä tietokannalla tarkoitetaan useimmiten SQL-kielistä tietokantatoteutusta.

## 5.1 Relaatiotietokannat

Relaatiotietokannat ovat nykyään yleisin tietokantojen muoto. Niiden perusta on DR. E.F.Coddin vuonna 1970 julkaisemassa artikkelissa ”A Relational Model of Data for Large Shared Data Banks”. Artikkelissa kuvataan malli tietokantojen hallintaan, jossa tiedot tallennetaan tauluihin, jotka ovat sukua toisilleen puumaisessa järjestelmässä. (5.) Uusi relaatiomalli korjaa aiempien järjestelmien ristiriitaisuuksia.

Tietokannan peruselementti on taulu, joka koostuu riveistä ja sarakkeista. Taululle merkitään perusavain, jonka tehtävä on erottaa tietueet toisistaan. Perusavaimen on oltava uniikki, eikä sen arvo voi olla tyhjä (NULL). Aina tämä ei ole mahdollista toteuttaa yhdellä sarakkeella, joten perusavain voi koostua myös kahdesta tai useammasta sarakkeesta. (5, s. 380.) Jos perusavaimeksi sopivaa saraketta tai sarakejoukkoa ei ole mahdollista löytää, luodaan surrogaatti (6). Surrogaatti on avainehdokkaan vaatimukset täyttävä uusi kenttä. Yleinen käytäntö on asettaa surrogaatiksi integer-tyyppinen muuttuja, jolla on AUTO INCREMENT-, NOT NULL- ja UNSIGNED-parametrit. Tämä tarkoittaa, että surrogaatti ei voi olla tyhjä, se on etumerkitön (saa vain positiivisia arvoja) ja se kasvaa automaattisesti, kun tauluun syötetään uusi tietue. (6.)

Taulut usein myös liittyvät toisiinsa äiti-lapsi-tyyppisesti. Tällöin äititaulun on oltava olemassa, ennen kuin lapsitaulua tai viiteavainta voidaan luoda. Viiteavain on lapsitaulun sarake, jonka tieto on yhteinen äititaulun perusavaimen kanssa. On yleistä, että viiteavain on samalla myös lapsitaulun perusavain tai osa sitä.

### 5.1.1 Yhteydet ja yhteystyypit

Yhteydellä tarkoitetaan kahden taulun välistä pysyvää liitosta, joka määritellään viiteavaimella toiseen tauluun. Yhteystyyppejä on kolmenlaisia: yhdestä yhteen, yhdestä moneen ja monesta moneen. (6.)

#### Yhdestä yhteen -yhteys

Yhdestä yhteen -yhteys on olemassa, jos taulun 1 yksittäinen tietue liittyy vain ja ainoastaan yhteen taulun 2 tietueeseen ja taulun 2 yksittäinen tietue liittyy ainoastaan yhteen taulun 1 tietueeseen. Tätä yhteystyyppiä ei suositella käytettäväksi usein. Sen sijaan, että loisi yhdestä yhteen -yhteyksiä, suositellaan tutkimaan mahdollisuutta yhdistää tauluja. (6.)

#### Yhdestä moneen -yhteys

Yhdestä moneen -yhteys tarkoittaa nimensä mukaan tilanteita, jolloin taulun 1 yksi tietue voi liittyä taulun 2 yhteen tai useampaan tietueeseen ja taulun 2 yksittäinen tietue liittyy ainoastaan yhteen taulun 1 tietueeseen. Tätä yhteystyyppiä suositellaan käytettäväksi, sillä se toteuttaa normaalimuotoa ja samalla pitää yhteen liittyviä tietoja samassa taulussa. (6.)

#### Monesta moneen -yhteys

Monesta moneen -yhteys tarkoittaa tilanteita, jolloin taulun 1 yksittäinen tietue voi liittyä yhteen tai useampaan taulun 2 tietueeseen ja taulun 2 yksittäinen tietue voi liittyä yhteen tai useampaan taulun 1 tietueeseen. Tämä yhteystyyppi on puhtaasti teoreettinen. Mikään nykyisin käytössä olevista järjestelmistä ei pysty toteuttamaan monesta moneen yhteyttä, koska se aiheuttaisi konfliktin tai silmukan taulujen tietueissa. (6.)

### 5.1.2 Normalisointi

Normalisointi on prosessi, jolla järjestetään tietokannan tietoja. Sen tarkoitus on lyhentää taulujen rakenteita, poistaa redundanssia, vakauttaa järjestelmää ja lisätä tietokannan joustavuutta. (7.) Normalisoinnilla varmistetaan hyvä suorituskyky ja luettavuus tietokannalle. Se suoritetaan kolmessa vaiheessa I–III.

Normalisointi tehdään erikseen jokaiselle taululle. Jos tietokannan kaikki taulut ovat kolmannessa normaalimuodossa, voidaan sanoa, että tietokanta on normalisoitu. Normaalimuotojen selittämiseen tarvitaan käsite funktionaalinen riippuvuus. (6.)

**Funktionaalinen riippuvuus**

Funktionaalista riippuvuutta merkitään A<-B, mikä tarkoittaa, että funktiojoukko A on funktionaalisesti riippuvainen funktiojoukko B:stä. Jos taulussa kahdella rivillä on samat arvot sarakejoukolla B, on niillä samat arvot myös sarakejoukko A:lla. (6.)

Täydellistä funktionaalista riippuvuutta voidaan kuvata seuraavalla tavalla:

Olkoon A, B ja C sarakkeita. A on täydellisesti riippuvainen B:stä ja C:stä, jos A ei ole yksistään riippuvainen B:stä tai C:stä, mutta on riippuvainen niiden yhteydestä. (6.)

Taulukko 1 on täysin normalisoimaton taulu ja se tulee olemaan pohjana normalisointiprosessin esimerkille.

*TAULUKKO 1. Normalisoimaton taulu*

| tyontekija_nro | esimies | esimiehen_huone | tyo1 | tyo2 | tyo3 |
|---|---|---|---|---|---|
| 2005 | Hietanen | 204 | varastointi | pakkaus | siivous |
| 4431 | Yli-Maunula | 311 | myyntireskontra | markkinointi | |

**I Normaalimuoto**

Taulu on ensimmäisessä normaalimuodossa, kun

- yksittäisissä tauluissa ei ole toistuvia attribuutteja
- toisiinsa liittyville tiedoille on luotu oma taulunsa (7)
- jokainen sarake on jakamaton kokonaisuus (6).

Taulukko 2 on ensimmäiseen normaalimuotoon muutettuna edellinen taulukko.

*TAULUKKO 2. Sama taulu ensimmäisessä normaalimuodossa*

| tyontekija_nro | esimies | esimiehen_huone | tyo |
|---|---|---|---|
| 2005 | Hietanen | 204 | varastointi |
| 4431 | Yli-Maunula | 311 | myyntireskontra |
| 2005 | Hietanen | 204 | pakkaus |
| 2005 | Hietanen | 204 | siivous |
| 4431 | Yli-Maunula | 311 | markkinointi |

**II Normaalimuoto**

Taulu on toisessa normaalimuodossa (Taulukko 3), kun

- taulu toteuttaa I normaalimuodon
- Luodaan oma taulu attribuuteille, joita tarvitaan useassa muussa taulussa (6).

*TAULUKKO 3. Työntekijä-taulu toisessa normaalimuodossa*

| tyontekija_nro | esimies | esimiehen_huone |
|---|---|---|
| 2005 | Hietanen | 204 |
| 4431 | Yli-Maunula | 311 |

Taulukko 4 on normalisoinnin takia Työntekijä-taulusta erotettu taulu, joka on nimetty Projekti-tauluksi.

*TAULUKKO 4. Projekti-taulu*

| tyontekija_nro | tyo |
|---|---|
| 2005 | varastointi |
| 4431 | myyntireskontra |
| 2005 | pakkaus |
| 2005 | siivous |
| 4431 | markkinointi |

### III Normaalimuoto

Taulu on kolmannessa normaalimuodossa, kun

- Taulu toteuttaa I ja II normaalimuodon
- Tauluista on poistettu tiedot, jotka eivät ole riippuvaisia avaimesta (7).

Taulukko 5 on kolmannen normaalimuodon kriteerit täyttävä Työntekijä taulu. Taulukko 6 on Hallinto-tauluksi nimetty Työntekijä-taulusta erotettu osa.

*TAULUKKO 5. Työntekijä-taulu kolmannessa normaalimuodossa*

| tyontekija_nro | esimies |
|---|---|
| 2005 | Hietanen |
| 4431 | Yli-Maunula |

*TAULUKKO 6. Hallinto-taulu*

| esimies | esimiehen_huone | jaosto |
|---|---|---|
| Hietanen | 204 | 12 |
| Yli-Maunula | 311 | 10 |

Esimerkkinä käytetty normalisoimaton taulu Työntekijä, on normalisoituna kolmen taulun kokoinen relaatiotietokanta.

### 5.2 SQL

SQL (Stuctured Query Language) suomeksi rakenteinen kyselykieli, mikä on hieman harhaanjohtava, sillä SQL:llä voi tehdä paljon enemmän kuin suorittaa ainoastaan kyselyjä tietokannasta. Tunnuksenomaista SQL-kielessä on komentojen kirjoitus isoin kirjaimin, jolloin käskysanat on helppo erottaa operandeista. SQL:n komennot voidaan jakaa viiteen eri luokkaan (6)

1. kyselylauseet
2. datanmanipulointilauseet
3. datanmäärittelylauseet
4. transaktioidenkontrollointilauseet
5. datankontrollointilauseet.

Datan- ja transaktioidenkontrollointilauseet eivät ole tämän työn kannalta oleellisia, joten rajaan niiden esittelyn pois tästä teoriasta. Jossakin toisessa tietokantoja koskevassa työssä ne voivat olla hyvinkin tärkeitä.

#### 5.2.1 Kyselylauseet

SQL-kyselylauseet hakevat tietokannasta hakuehdot täyttävää tietoa. Kyselylause alkaa aina merkityllä sanalla SELECT. Sillä voidaan hakea rajattuja attribuutteja taulusta tai rajaamattomia käyttäen *-merkkiä.  Parikseen SELECT-lause tarvitsee kohteen, joka määritellään FROM-sanaa käyttäen. Esimerkiksi haetaan kaikkia attribuutteja taulusta Projekti.

**SELECT * FROM Projekti;**

Jos haetaan vain tiettyjä attribuutteja, erotellaan attribuutit pilkuin SELECT-sanan jälkeen.

**SELECT tyontekija_nro, tyo FROM Projekti;**

Jotta voidaan rajata hakuja koskemaan vain määriteltyjä tietueita, käytetään WHERE-sanaa apuna. Sen syntaksi on hyvin lähellä C-kielestä tuttua if-rakennetta. Esimerkiksi haetaan samasta taulusta vain yhden henkilön projektitietoja.

**SELECT * FROM Projekti WHERE tyontekija_nro=2005;**

Seuraavaksi haetaan kaikkia projektitietoja, joissa tämä työtekijä ei ole ollut osallisena.

**SELECT * FROM Projekti WHERE tyontekija_nro<>2005;**

Kyselyihin liittyvät oleellisesti myös erilaiset koostefunktiot, joita ovat (6)

- Maksimi MAX( )
- Minimi MIN( )
- Summa SUM( )
- Määrä COUNT( )
- Keskiarvo AVG( ).

Koostefunktioita käytetään yleensä GROUP BY -komennon kanssa, jolloin voidaan yhdistää tietyn attribuutin mukaan kaikki toistuvat tietueet ja ryhmitellä koostefunktio toisistaan eroaville tietueille. GROUP BY -komennolle voidaan määrittää, onko haettava tieto laskevassa vai nousevassa aakkos- ja numerojärjestyksessä merkityillä sanoilla ASC tai DESC. (8.) Esimerkiksi kootaan työntekijöiden palkat ryhmiteltynä työntekijänumeron mukaan kasvavassa järjestyksessä. Tähän on otettu mukaan myös AS-sana, joka kirjoittaa attribuutille halutun otsikon korvaten metodin nimen otsikkokentästä.

**SELECT tyontekija_nro, SUM(palkka) AS 'kokonaispalkka'**

**FROM Palkat GROUP BY tyontekija_nro ASC;**

**Liitos ja alikysely**

Usein tietokantakyselyjä tehdessä huomataan, että pelkän yhden taulun attribuutit eivät ole riittäviä. Esimerkiksi tarvitaan esimiehen huoneen numero tulostaa toisesta taulusta ja työntekijän numero toisesta. Tätä varten SQL:ssä on määritelty sisäisiä ja ulkoisia liitoksia.

**SELECT tyontekija_nro, Tyontekija.esimies, esimiehen_huone**

**FROM Tyontekija INNER JOIN Hallinto**

**ON Tyontekija.esimies=Hallinto.esimies;**

Edellisessä esimerkissä käytetään sisäliitosta, joka näyttää tauluista yhteiset tietueet. Tauluilla, joiden välille ollaan luomassa liitosta, on oltava relaatio. Jotta konfliktia ei synny päällekkäisten attribuuttien kanssa, on määriteltävä, kummasta taulusta attribuutti haetaan. Edellä määrittely on tehty attribuutille esimies. Lisäksi liitokselle on määriteltävä, mistä attribuuteista taulut liitetään ON-lauseella. Jos Työntekijä-taulussa olisi tietueita, joita ei esimies taulusta löydy ja ne haluttaisiin myös ottaa hakuun mukaan, käytettäisiin ulkoista liitosta. Ulkoista liitosta merkitsevä sana viittaa aina vasemmalle tai oikealle puolelle, RIGHT tai LEFT JOIN. Suunta viittaa aina siihen tauluun, josta halutaan ottaa kaikki tietueet mukaan hakuun.

**SELECT tyontekija_nro, Tyontekija.esimies, esimiehen_huone**

**FROM Tyontekija LEFT JOIN Hallinto**

**ON Tyontekija.esimies=Hallinto.esimies;**

Alikyselyä voidaan käyttää, kun on haettava tietoa taulusta, jonka ehtona tarvitaan toisen taulun tietoa. Alikysely on normaalimuotoinen kyselylause, joka sijoitetaan haettavan tiedon WHERE-ehtoon. Esimerkiksi haetaan Projekti-taulusta kaikkia töitä, joissa Hietanen on ollut esimiehenä työntekijöille.

**SELECT tyo FROM Projekti**

**WHERE tyontekija_nro=(**

**SELECT tyontekija_nro FROM Tyontekija**

**WHERE esimies='Hietanen');**

### 5.2.2 Datan manipulointilauseet

Datan manipulointi (DML) mahdollistaa sisällön muuttamisen tauluissa. Sisältöä muutetaan INSERT-, UPDATE- ja DELETE-lauseiden avulla (6).

Uuden tietueen syöttämistä varten käytetään INSERT-lausetta. Jotta tietue kohdistuu oikeaan tauluun, käytetään merkittyä sanaa INTO taulun nimen kanssa. Seuraavaksi merkitään sulkujen sisään pilkuin eroteltuina attribuutit, joihin tietueen data kohdistuu. Asetettava data syötetään itse syötettyjen attribuuttien määräämässä järjestyksessä VALUES-sanan jälkeen. Kirjallinen data erotetaan puolilainausmerkein, numeerinen voidaan kirjoittaa ilman. Esimerkissä syötetään Hallinto-tauluun uusi esimies ja hänen työhuoneensa.

**INSERT INTO Hallinto (esimies, esimiehen_huone)**

**VALUES ('Lehtinen', 214);**

Ainoastaan tilanteissa, joissa kaikkiin taulun attribuutteihin tulee dataa, voidaan jättää kirjoittamatta attribuuttien nimet ennen määrättyä sanaa VALUES. Tällöin data on syötettävä samassa järjestyksessä kuin taulussa on attribuutit.

**INSERT INTO Hallinto VALUES ('Sipola', 111, 8);**

Kun tietokannan tietuetta tarvitsee muuttaa tai päivittää, käytetään UPDATE-lausetta. Käsky alkaa aina UPDATE-sanalla, eikä sille tarvitse erikseen kirjoittaa toista sanaa osoittamaan taulua, vaan taulu merkitään heti UPDATE-sanan jälkeen. SET-sanalla määritetään, mitä arvoja muutetaan. Lisäksi WHERE-sanalla määritetään, mitä tietueita päivitetään.

**UPDATE Hallinto SET jaosto=12 WHERE esimies='Lehtinen';**

Jos WHERE-määritelmän jättää käyttämättä, attribuutti päivittyy kaikille tietueille taulussa, mikä yleensä ei ole toivottavaa. Varsinkaan, koska edelliset arvot eivät ole palautettavissa, ellei päivitystä ole suorittanut esimerkiksi osana transaktiota tai SQL serverillä ei ole automaattista varmuuskopiointia käytössä.

Tietueiden poistaminen tietokannasta tapahtuu DELETE-lauseen avulla. DELETE käyttää FROM-sanaa merkitsemään taulua, josta tietue poistetaan. Kuten UPDATE-lauseen kanssa, on hyvin tärkeää muistaa käyttää WHERE-määritystä, koska muuten tulee poistaneeksi kaikki tietueet taulusta. Seuraava esimerkki poistaa kaikki tietueet taulusta Hallinto, missä esimies-attribuutin arvo on Hietanen.

**DELETE FROM Hallinto WHERE esimies='Hietanen';**

On myös tilanteita, jolloin on hyvä tyhjentää koko taulu, joka tapahtuu jättämällä WHERE-määritelmä pois.

**DELETE FROM Hallinto;**

### 5.2.3 Datan määrittelylauseet

Tietokannan rakenteita koskevat käskyt kuuluvat datan määrittelylauseisiin (DDL). Niistä tärkeimmät ovat taulujen luomisen ja muuttamisen työkalut. Myös taulujen ja niiden rakenteiden poistamisen työkalut ovat tärkeitä.

**CREATE**

CREATE on komento jolla voidaan luoda uusia tauluja. Se ottaa vastaan taulun nimen, attribuuttien nimet, attribuuttien tyypit, perusavaimen, mahdolliset viiteavaimet, viite-eheyssäännöt ja tietokantamoottorin määritelmän. seuraavassa esimerkissä luodaan yksinkertainen taulu, jolla on vain kaksi attribuuttia, eikä sillä ole viiteavaimia.

**CREATE TABLE Kaupungit(**

**postinro CHAR(5) PRIMARY KEY,**

**kaupunki VARCHAR(20)) Engine=InnoDB;**

Edellinen taulu olkoon isätaulu Asiakkaat-taululle, joka on hieman laajempi kuin Kaupungit-taulu. Perusavaimena asiakastaululle on asiakasnumero, joka on etumerkitön kokonaisluku. Asiakasnro kasvaa automaattisesti, kun tauluun syötetään uusi tietue, eikä sen arvo voi olla tyhjä. Asiakkaat taululla on yksi viiteavain, postinro, joka viittaa Kaupungit-taulun postinro attribuuttiin. Kun viiteavaimia luodaan, on pidettävä huoli, että attribuuttien tyypit vastaavat toisiaan. Viite-eheydestä on myös huolehdittava.

**CREATE TABLE Asiakas(**

**asiakasnro INT PRIMARY KEY AUTO_INCREMENT UNSIGNED NOT NULL,**

**etunimi VARCHAR(20),**

**sukunimi VARCHAR(20),**

**ika TINYINT,**

**sposti VARCHAR(45),**

**katuosoite VARCHAR(45),**

**postinro CHAR(5),**

**FOREIGN KEY (postinro) REFERENCES Kaupungit(postinro)**

**ON UPDATE CASCADE**

**ON DELETE RESTRICT) Engine=InnoDB;**

**Viite-eheys**

Viite-eheydellä pyritään estämään orpojen tietueiden syntyminen. Orvot tietueet ovat tietueita, joilla on ollut viittaus äititietueeseen, mutta ovat menettäneet sen äititietueen päivityksen tai poiston vaikutuksesta. Viite-eheydestä huolehditaan viite-eheyssäännöin. Niitä ovat

- RESTRICT
- CASCADE
- SET NULL
- SET DEFAULT.

Viite-eheyssäännöt määritetään luotaessa taulua ja ne koskevat kahta tilannetta: kun isätietuetta päivitetään tai kun se poistetaan. RESTRICT estää toiminnon ja antaa virheilmoituksen. CASCADE vyöryttää muutoksen lapsitietueille automaattisesti. SET NULL asettaa lapsitietueiden lapsiattribuutit arvoon NULL. SET DEFAULT asettaa lapsitietueiden lapsiattribuutit johonkin ennalta määrättyyn arvoon. Taulun tyypistä ja tarkoituksesta riippuu, mitä viite-eheyssääntöjä on käytettävä.

**ALTER**

Taulujen rakennetta voidaan muuttaa ALTER-lauseen avulla. Kun taulu on luotu, voidaan sitä hallita monipuolisesti. Tauluja voi jopa muokata silloinkin, kun niissä on jo dataa tallennettuna. Silloin kannattaa pitää attribuutteja muokatessa huoli, ettei dataa häviä muutosten yhteydessä. Hyvä tapa on tehdä taulusta kopio sisältöineen ja muokata kopiota. Datahävikki on riski, jos varmuuskopiota ei ole otettu. Yleistä on että tietokannan luonnin jälkeen johonkin tauluun halutaan lisätä attribuutteja, joita ei ollut aiemmin tarvittu. Tauluun lisätään attribuutteja komennolla ADD COLUMN.

**ALTER TABLE Kaupungit ADD COLUMN Laani VARCHAR(20);**

Samalla tavalla voidaan taulusta poistaa attribuutteja DROP-komennolla.

**ALTER TABLE Kaupungit DROP Laani;**

Avaimia voi myös lisätä tai poistaa ADD- tai DROP-komennoin.

**ALTER TABLE Kaupungit DROP PRIMARY KEY:**

**ALTER TABLE Kaupungit ADD PRIMARY KEY (postinro, kaupunki);**

**DROP**

Aivan kuten tauluista voidaan poistaa attribuutteja DROP-komennolla, voidaan koko taulu poistaa, kun käytetään DROP-lausetta. Se on hyvin vahva ja tietokannan hallinnoijan on hyvä olla täysin perillä mitä tekemässä, sillä taulujen poisto ei varoita sen tehokkuudesta mitenkään. Kun taulu poistetaan, sen sisältämä data poistuu myös. Jos taulun tietueilla on lapsitietueita, menetellään viiteeheyssääntöjen mukaisesti.

**DROP TABLE Asiakas;**

# 6 PHP-OHJELMOINTIKIELI

PHP (Hypertext Preprocessor) on web-ohjelmointikieli, jonka kehityksen aloitti Rasmus Lerdorf vuonna 1994 (8, s. xi). PHP on niin sanottu skriptikieli, joka tulkataan vasta suoritettaessa. Vasta-aloittelijoille PHP on helposti ja nopeasti lähestyttävä kieli. Esimerkiksi aloittelevan sovelluskehittäjän ei tarvitse huolehtia muuttujatyypeistä, sillä PHP tulkitsee ne muuttujien sisältöjen mukaan. Ammattilaisille PHP on hyvin nopea kieli kehittää web-sovelluksia. PHP voidaan sisällyttää HTML-dokumenttien sisään tai sillä voidaan tuottaa hypertekstiä. (9, s. 12.)

## 6.1 Muuttujat

PHP-kielessä ei ole tarvetta määritellä muuttujia etukäteen, joitakin poikkeuksia lukuun ottamatta. Muuttujan tyyppi määräytyy sen sisältämän datan mukaan ja voi vaihtua kesken ajon automaattisesti. Muuttujiin viitataan aina dollarimerkillä ($) muuttujan nimen alussa, esimerkiksi $temp. Muuttujanimissä voidaan käyttää merkkejä a–z, A–Z, _, 0–9 ja ascii-merkkejä välillä 127–255. Myös skandinaavisia ä- ja ö-kirjaimia voidaan käyttää, mutta se voi johtaa ongelmiin joissakin ohjelmissa. Muuttujan nimi ei saa alkaa numerolla. (9, s.62.) Muuttujan paikasta riippuu, käsitelläänkö sitä globaalina vai paikallisena, aivan kuin C-kielessä. Paikalliset ovat käytössä vain siinä ohjelmalohkossa, missä ne ovat esitelty. Globaalit esitellään pääohjelmalohkossa. Funktiot näkevät globaalit muuttujat ainoastaan, jos ne määritellään kyseisen funktion sisällä avainsanalla global. Muussa tapauksessa funktio luo uuden paikallisen muuttujan samalla nimellä ja käyttää sen arvoa globaalin sijasta. (9, s. 63.) Staattiset muuttujat ovat paikallisia muuttujia, jotka eivät menetä arvoaan funktiosta poistuttaessa. Ne määritellään avainsanalla static funktion sisällä. (11.)

## 6.2 Muuttujien vastaanotto PHP-dokumentissa

Kuten aiemmin todettu, PHP on web-ohjelmointikieli. Sitä voidaan käyttää esimerkiksi datan tallentamiseen tietokantaan. PHP-dokumentin on siis pystyttävä ottamaan vastaan argumentteja. Tätä varten PHP-kieleen on sisällytetty kaksi metodia: GET ja POST.

**GET-metodi**

Kun PHP-dokumenttia kutsuttaan selaimesta, voidaan URL:in loppuun sisällyttää datapareja, joissa esitellään nimi ja muuttuja. Jos datapareja on useita, voidaan ne erottaa et-merkein (&). Muuttuja voidaan sitten ottaa talteen PHP-dokumentissa $_GET()-metodilla. Esimerkiksi kutsutaan verkkosivua osoitteessa

**www.esimerkki.fi/esim.php?kaupunki=Kotka&postinro=48100.**

Esim.php voi ottaa vastaan nämä kaksi muuttuja seuraavalla tavalla:

```
<?php

$kaupunki = $_GET[’kaupunki’];

$postinro = $_GET[’postinro’];

echo $postinro . ” ” . $kaupunki;

?>.
```

Verkkosivulle ilmestyy tällöin ainoastaan teksti

**48100 Kotka.**

GET metodi on helppokäyttöinen ja sopii hyvin käytettäväksi pienten muuttujien siirtämiseen, mitkä eivät ole edes osin salaisia. Jos GET ei löydä muuttujaa, sen arvoksi jää NULL. Jos muuttujan pitäisi sisältää et-merkki tai välilyönti, joudutaan URL koodaamaan, ennen kuin sen voi syöttää selaimeen. Ilman koodausta PHP ymmärtää muuttujan ensimmäiseen välilyöntiin asti, mutta jättää loput tiedot huomiotta. Tietosuojan kannalta, jos tarvitsee lähettää salaisia lukuja tai muuta tietoa, joka ei kuulu loppukäyttäjän silmille, ei ole hyvä käyttää GET-metodia. Selaimessa kutsuttaessa nämä kutsut jäävät sivuhistoriaan muuttujineen ja muuttujanimineen. GET-metodi suorastaan kutsuu väärinkäytöksiin. Sen sijaan on hyvä käyttää POST-metodia.

**POST-metodi**

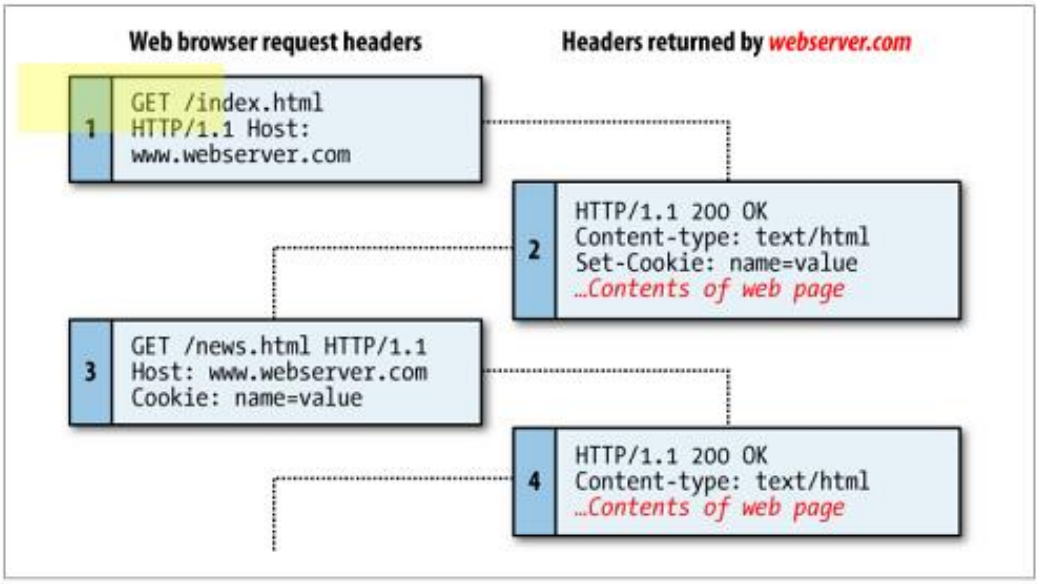
POST-metodia voidaan käyttää verkkosivujen välillä tiedonsiirtoon tai ohjelmakutsussa, mutta silloin ohjelman on pystyttävä lähettämään POST-tyyppisesti dataa. Tieto siirtyy HTML-otsakkeessa. Sen suojauksesta vastaa HTTP-protokolla. Käyttämällä secure HTTP:tä voidaan olla varmoja, että tieto siirretään turvallisesti (12). Tieto otetaan vastaan seuraavasti:

```
<?php

$kaupunki = $_POST['kaupunki'];

$postinro = $_POST['postinro'];

echo $postinro . " " . $kaupunki;

?>
```

Tässä opinnäytteessä käsitellään tiedon lähetys, POST-metodia hyväksikäyttäen Qt-ohjelmointikielellä, kappaleessa 6.2.

### 6.3 Evästeet

Evästeet ovat HTTP-otsakkeena selaimelle tallennettava merkkijono, joka pitää sisällään verkkosivun suoritukseen tarvittavaa tietoa (9, s. 240–245). Ne voivat sisältää lähes mitä tahansa kirjain- tai numerotyyppistä dataa, mutta korkeintaan 4 kilotavun verran. Yleisiä tapoja käyttää evästeitä, ovat istuntojen ylläpito, datan säilytys uudelleen sivustolle palattaessa, ostoskorien sisällön säilyttäminen, kirjautumistunnusten säilyttäminen ja moni muu palvelu, joka hyötyy datan säilyttämisestä (13, s. 279). Evästeet mahdollistavat sisäänkirjautuneena pysymisen verkkosivulla, kun käyttäjä selailee muita verkkosivuja. Jopa selaimen sulkeminen ja uudelleenkäynnistäminen on mahdollista menettämättä istuntoaan verkkosivustossa. Evästeen pystyy lukemaan ainoastaan domainista, josta se on lähetetty (13, s. 279). Kuva 3 kertoo evästeen tallennukseen liittyvästä keskustelusta palvelimen ja selaimen välillä.




*KUVA 3. Evästeen tallennus selaimelle (13. s.280)*

Evästeet kulkeutuvat palvelimelta selaimelle HTTP-otsakkeessa jo ennen kuin sivuston sisältö on siirtynyt ollenkaan. Kuva 3 selventää selaimen ja palvelimen välistä keskustelua, kun selain pyytää aukaista verkko-osoitteen ja palvelin siirtää sille evästeen. Ensimmäisessä vaiheessa selain pyytää tiedoston ja tarkentaa kolmannella rivillä, miltä palvelimelta. Toisessa vaiheessa palvelin ottaa vastaan pyynnön ja lähettää vastauksena, minkä tyyppistä tiedostoa selaimen tulee lukea ja lisäksi evästeen. Kun selain on ottanut evästeen vastaan, palauttaa se evästeen aina samalle palvelimelle niin kauan, kunnes eväste poistetaan tai sen elinaika päättyy.

PHP-kielellä evästeen luonti on hyvin yksinkertaista, on vain käytettävä setcookie()-metodia. Setcookie() ottaa vastaan parametreja seuraavassa järjestyksessä:

1. nimi
2. arvo
3. voimassaoloaika
4. URL:n hakupolku
5. URL:n verkkoalue
6. turvallisuusparametri
7. httponly parametri.

Esimerkiksi luodaan eväste, joka pitää sisällään käyttäjätunnuksen verkkopalveluun. Se on voimassa viikon luonnista lähtien.

**setcookie(’username’, ’Player1’, time() + 60 * 60 * 24* 7, ’/’);**

Taulukko 7 selventää oikeassa järjestyksessä, mihin evästeelle annettavat parametrit vaikuttavat.

*TAULUKKO 7. Evästeiden parametrit (13. s.281)*

| Parameter | Description | Example |
|---|---|---|
| `name` | The name of the cookie. This is the name that your server will use to access the cookie on subsequent browser requests. | `username` |
| `value` | The value of the cookie, or the cookie's contents. This can contain up to 4 KB of alphanumeric text. | `Hannah` |
| `expire` | (*optional*) Unix timestamp of the expiration date. Generally, you will probably use `time()` plus a number of seconds. If not set, the cookie expires when the browser closes. | `time() + 2592000` |
| `path` | (*optional*) The path of the cookie on the server. If this is a / (forward slash), the cookie is available over the entire domain, such as *www.webserver.com*. If it is a subdirectory, the cookie is available only within that subdirectory. The default is the current directory that the cookie is being set in and this is the setting you will normally use. | / |
| `domain` | (*optional*) The Internet domain of the cookie. If this is *webserver.com*, the cookie is available to all of *webserver.com* and its subdomains, such as *www.webserver.com* and *images.webserver.com*. If it is *images.webserver.com*, the cookie is available only to *images.webserver.com* and its subdomains such as *sub.images.webserver.com*, but not, say, to *www.webserver.com*. | `.webserver.com` |
| `secure` | (*optional*) Whether the cookie must use a secure connection (*https://*). If this value is `TRUE`, the cookie can be transferred only across a secure connection. The default is `FALSE`. | `FALSE` |
| `httponly` | (*optional*; implemented since PHP version 5.2.0) Whether the cookie must use the HTTP protocol. If this value is `TRUE`, scripting languages such as JavaScript cannot access the cookie. (Not supported in all browsers). The default is `FALSE`. | `FALSE` |

Evästeen lukeminen onnistuu COOKIE-metodilla. Sen syntaksi on sama kuin POST- tai GET-metodeilla. Esimerkkinä luetaan edellisessä esimerkissä luotu eväste username. Isset() tarkastaa, onko username-evästettä olemassa ja ettei sen arvo ole NULL.

**if(isset($_COOKIE[’username’])) $kayttaja = $_COOKIE[’username’];**

Evästeen tuhoamiseen on kaksi hyvää tapaa. Ensimmäinen perustuu siihen, että sen voimassaoloaika asetetaan menneisyyteen, jolloin palvelin tuhoaa evästeen. Voimassaoloaikaa lukuun ottamatta kaikkien parametrien on vastattava alkuperistä evästettä, muuten poisto epäonnistuu. (13, s.282).

**setcookie(’username’, ’Player1’, time() - 2592000, ’/’);**

Toinen tapa on luoda eväste, jolle ei esitetä arvoa. Selain ei tämän jälkeen lähetä kyseistä evästettä (10, s.242). Evästeen nimen on vastattava poistettavaa evästettä.

**setcookie(’username’);**

### 6.4 MySQL ja PHP

PHP:lla on joukko hyviä työkaluja MySQL-tietokannan käyttöön. Tässä kappaleessa esitellään, miten tietokantaan tallennetaan uusia tietueita sekä miten niitä luetaan.

Ensimmäinen askel kohti tietokannan lukemista, on luoda yhteys tietokantaan funktiolla mysql_connect(). Tämä on vanhentuva metodi, jonka sijasta tulisi käyttää mysqli::__construct()-metodia (14). Työtä tehdessä tällaista merkintää ei vielä ollut, joten esittelen metodin toteutetulla tavalla. Se ottaa parametreinä vastaan palvelimen, käyttäjänimen, salasanan, uuden linkin ja asiakkaan liput. Kaksi viimeistä parametria eivät ole pakollisia yhteyden luomisen kannalta. Palvelimeksi valitaan localhost, jos PHP-tiedosto on samalla palvelimella kuin tietokanta. Palvelin-parametriin voidaan sisällyttää portti esim. ”hostname:port” tai polku paikalliseen socketiin. (15.) Normaalisti, jos uudelleen kutsutaan mysql_connect()-metodia samoilla argumenteilla, uutta linkkiä ei luoda, vaan metodi palauttaa tunnisteen jo avoinna olevaan yhteyteen. Uusia yhteyksiä voi kuitenkin avata samoilla argumenteilla, jos metodin kutsussa määrittelee uusi linkki -parametrin aktiiviseksi. Asiakkaan lippuparametri on tarkoitettu ainoastaan käytettäväksi, jos ylimääräinen laajennus on lisätty PHP:hen. Sillä ei ole merkitystä tämän opinnäytteen kannalta. Tässä esimerkissä luodaan yhteys paikalliseen tietokantaan esimerkkitunnuksilla. Mysql_error() palauttaa virheen,

jos sellainen tapahtuu yhteyden muodostuksessa. Mysql_close() sulkee yhteyden.

```
$connection = mysql_connect('localhost', 'kayttajatunnus', 'salasana');

if(!$connection){

        die('Ei luotu yhteyttä: ' . mysql_error());

}

echo 'Yhteys luotu onnistuneesti';

mysql_close($connection);
```

Seuraavaksi on valittava tietokanta, uudemmassa metodissa tämä on sisällytetty yhteyden luomiseen. Se onnistuu mysql_select_db()-metodilla. Sen parametreja ovat tietokanta ja yhteyden tunnus. Jos tunnusta ei merkitse, viimeistä avattua yhteyttä käytetään tunnuksena. (16.)

```
$db = mysql_select_db('tietokanta1', $connection);

if (!$db) {

        die('Ei voi kayttaa tietokanta1 : ' . mysql_error());

}
```

Kun yhteys on luotu ja tietokanta valittu, saadaan luettua tietokannasta haluttua tietoa kyselylauseiden avulla. Kyselylause sijoitetaan muuttujaan, joka annetaan parametrina mysql_query()-metodille. Toisella parametrilla voidaan määrittää yhteys. Oletuksena käytetään viimeistä avattua yhteyttä. Kyselylause palauttaa resurssin onnistuessaan ja boolean-arvo FALSE:n epäonnistuessaan. Paluuarvon lukemiseen käytetään mysql_fetch_array()-metodia. (17.)
mysql_fetch_array() ottaa vastaan mysql_query()-metodilta tulleen paluuarvon ja palauttaa sieltä rivejä yksi kerrallaan array-muodossa. Useimmiten tätä käytetään while-silmukkaa hyväksikäyttäen. Silmukka on voimassa niin kauan, kuin mysql_fetch_array() palauttaa rivejä. Seuraavassa esimerkissä näytetään tämä

toiminnassa aiemmin käytössä olleelle Työntekijä-taululle. Row arrayn sisältöä kutsutaan taulussa olleilla attribuutin nimillä. On myös mahdollista käyttää nollasta alkavia järjestysnumeroita.

```
$query = ”SELECT * FROM Tyontekija”;

$result = mysql_query($query);

if(!$result) {

        die(’Virhe kyselylauseessa: ’ . mysql_error());

}

while ($row = mysql_fetch_array($result)) {

echo $row[’tyontekija_nro’];

echo $row[’esimies’];

}
```

Tietokantaan tallennus onnistuu myös mysql_query()-metodilla. Siinä tapauksessa ei tarvitse erikseen käsitellä paluuarvoa toisilla metodeilla, riittää kun ottaa virheet vastaan, jos niitä syntyy. $query-muuttujasta on kiinni tietokannan käyttö. SQL-kieltä käsitellään luvussa 4.2.

# 7 QT-OHJELMOINTIKIELI

Qt on ohjelmointikieli, joka aluksi kehitettiin C++-kehittäjien avuksi, mutta sittemmin kantavaksi ajatukseksi on muodostunut vahvan alustariippumattoman ohjelmakehitysympäristön tarjoaminen sitä tarvitseville. Samaa Qt-ohjelmaa voi käyttää Linux, Unix, Symbian, Meego, Windows, Mac ja muilla alustoilla. (18, s. 4.) Qt-kieltä käytetään tässä opinnäytteessä puhtaasti siitä syystä, että alkuperäinen Mypose-sovellus on kehitetty Qt:llä.

## 7.1 Signaalit ja slotit

signaalit ja slotit ovat tehokas tapa kommunikoida olioiden välillä. Signaalit ovat metodeja, joita ei suoriteta vaan lähetetään (18, s. 13). Signaalin lähettämiseen käytetään merkittyä sanaa emit. Signaali määritellään header-tiedostossa ja sen mukana voidaan lähettää dataa. Slotit ovat metodeita, jotka voidaan suorittaa signaalin herätteestä (18, s.13). Ne voivat olla julkisia, suojattuja tai yksityisiä aivan kuin mitkä tahansa metodit. Ei ole rajoitusta kuinka monta signaalia voidaan yhdistää yhteen slotiin tai kuinka moneen slotiin yksi signaali voidaan yhdistää. Signaalit välittyvät yleiseen tilaan, joten signaalin lähettävä luokka, ei ole tietoinen mikä luokka ja mikä slot ottaa sen vastaan. Seuraava esimerkki näyttää hyvin lyhyesti miten yhden luokan signaali voidaan yhdistää saman luokan slotiin. SomeMethod() lähettää signaalin, jossa on QString tyyppinen muuttuja mukana. OneSlot(QSting print) ottaa sen vastaan, kun yhteys on luotu connect-lauseella. OneSignal() siis lähetetään kahdesti, joista toinen kerta tulostaa lauseen debug-monitoriin, koska vasta silloin on luotu yhteys signaalin ja slotin välille.

**esim.h**

```
class esim : public QObject
{
//normal class declarations here.
private:
   void someMethod();
private slots:
   void oneSlot( QSting print );
signals:
   void oneSignal( QString );
```

```
}
```

**esim.cpp**

```
esim::esim(QObject* parent):QObject(parent)
{
    someMethod( );  //yhteyttä ei ole vielä luotu
    connect(this, SIGNAL(oneSignal(QString)), this,
SLOT(oneSlot(QString)));
    someMethod( ); //yhteys on luotu
}
void esim::oneSlot( QString print)
{
    qDebug() << "Signaalilta tullut teksti: " << print;
}
void esim::someMethod( )
{
   QString str = "Olen signaali";
   emit oneSignal( str );
}
```

Connect-lause on hyvin yksinkertainen, se ottaa vastaan neljä argumenttia

1. signaalin lähettävä olio
2. signaali
3. olio johon signaali yhdistetään
4. slot.

Huomioitavaa on että kääntäjä ei anna virhettä, jos connect-lauseen olioista ei ole vielä instansseja tai signaalien ja slotien tyypit eivät täsmää. Kääntäminen onnistuu, mutta ohjelma ei toimi halutulla tavalla. On siis oltava tarkkana yhteyttä luotaessa, että oliot ovat jo olemassa.

### 7.2 Verkkopyynnöt ja -vastaukset

QtNetwork module on kokoelma luokkia, jotka tarjoavat työkaluja verkkopohjaisten ohjelmien rakentamiselle. QNetworkAccessManager on yksi näistä luokista, sen avulla voidaan tehdä pyyntöjä verkkoon. Sillä on kolme meitä kiinnostavaa metodia put(), post() ja get().

Put() on tarkoitettu asioiden lataamiseksi verkkoon. Se ottaa vastaan QNetworkRequestin, jota useasti käytetään vain URL:n kirjoittamiseen. Lisäksi se ottaa

vastaan dataa joko QIODevice-, QHttpMultiPart- tai QByteArray-tyyppisenä ja siirtää sen QNetworkRequestin määräämään osoitteeseen.

Post() lähettää HTTP POST -pyynnön QNetworkRequestin määrittelemälle osoitteelle. Post-tyyppisesti datan vastaanotto PHP-kielessä käsiteltiin kappaleessa 5.2. Kuten put(), myös post() siirtää datan palvelimelle. Seuraavassa esimerkissä näytetään, miten tällainen pyyntö voisi rakentua.

```
QNetworkAccessManager *manager = new QNetworkAccess-
Manager(this);
QNetworkRequest req;
req.setUrl(QUrl("http://www.esimerkki.fi/esim.php"));

QUrl params;
params.addQueryItem("kaupunki", "Oulu");
params.addQueryItem("postinro", "90150");
```

Esimerkissä QUrl params hoitaa aluksi post-datan vastaanottamisen ja koodaamisen, jotta välilyönnit ja kaikki erikoismerkit toimivat siirrettävässä datassa. Siitä saatu tulos sijoitetaan QByteArray bArray:hyn, joka annetaan parametrina QnetworkAccesmanagerin post-metodille. Post() tarvitsee myös osoitteen, johon pyyntö lähetetään. Se annetaan QNetworkRequestia hyväksikäyttäen. On aina hyvä tarkkailla onnistuuko verkkopyynnöt ja -siirrot. Sitä varten managerin signaali finished yhdistetään QNetworkReplyä käsittelevään slotiin. Slotin toiminta riippuu ohjelman tarkoituksesta, mutta yleensä siihen ainakin sisällytetään vastauksen lukeminen ja mahdollisten virheiden tulostaminen. Esimerkiksi näin:

```
void esim::replyFinished(QNetworkReply* reply)
{
    QString answer = QString::fromUtf8(reply->readAll());
    qDebug() << "Reply: " << answer;

    if(reply->error() == QNetworkReply::NoError)
    {
        qDebug() << "Request finished without errors";
    }

    else
```

```
    {
        qDebug() << "Network Error: " << reply->error() <<
": " << reply->errorString();
    }

    reply->deleteLater();
}
```

Get()-metodi lähettää pyynnön palvelimelle saada ladattua QNetworkrequestin määrittelemä verkkosivu.

### 7.3 Linux system-komennot

QProcess-luokan avulla voidaan käynnistää Qt:n ulkopuolisia ohjelmia ja kommunikoida niiden kanssa (19). Prosessin käynnistämiseksi on QProcess::start()-metodille annettava ohjelman nimi tai polku ja komennot käytettävän komentokielen mukaan. Ohjelman nimi annetaan QString-muotoisena ja komennot QStringList-muodossa. Seuraava esimerkki tulostaa Linuxilla koko PATH-muuttujan. Ohjelmana käytetään Linuxin Shelliä.

```
QString result = "";
QString prog = "/bin/bash"; //ohjelman nimi
QStringList arguments; //komennot
arguments << "-c" << "echo $PATH";
QProcess* process = new QProcess(); //luodaan process olio
process->start(prog , arguments); //ohjelman käynnistys
process->waitForFinished(-1); //odotetaan prosessin suori-
tusta
result = process->readAll(); //luetaan saatava data
if(result.length() > 0)
{
    result.chop(1); //leikataan viimeinen merkki pois, se
ei kuulu PATH-muuttujaan.
    qDebug() << "PATH: " << result;
}
else
```

```
{
    qDebug() << "Couldn't find system variable PATH";
}
delete process;     //tuhotaan process olio
```

QProcess toimii asynkronisesti, siksi on odotettava vastauksen valmistumista waitForFinished()-metodilla. Se odottaa niin kauan, että prosessi on valmis, tai kunnes parametrina annettu aika millisekunteina täyttyy. Jos sille annetaan arvo -1, ei timeout ole käytössä ja odotetaan vain valmistumista. Jos odotus tehdään pääsäikeessä, voi se aiheuttaa ohjelman jäätymisen. (19.)

### 7.4 Tiedoston lukeminen ja siihen kirjoittaminen

QFile-luokka antaa mahdollisuuden kommunikoida tiedostojen kanssa kesken ohjelman suorittamisen. Lukeminen tapahtuu aukaisemalla tiedosto QFile::open()-metodilla ja sijoittamalla sen jälkeen QFile-olio QTextStream-olioon. QTextStream-oliosta voidaan lukea joko koko tiedosto readAll()-metodilla, tai yk-si rivi kerrallaan readLine()-metodilla. Esimerkkinä readFile()-metodi, joka ottaa vastaan tiedostonimen ja palauttaa QStringListina tiedoston sisällön. QStringListin alkioihin sijoitetaan luetut rivit siinä järjestyksessä, jossa ne ovat alkuperäisessä tiedostossa.

```
QStringList esim::readFile(QString name)
{
    QFile *file = new QFile(this); //luodaan file olio
    QString filename(QApplication::applicationDirPath() +
QDir::separator() + name);
    file.setFileName(filename); //annetaan tiedoston polku
    file.open(QIODevice::ReadWrite); //aukaistaan tiedosto
    QStringList content;
    QTextStream in(file); //luodaan in olio

    while(!in.atEnd()) //käydään tiedosto läpi rivi riviltä
    {
```

```
        content.append(in.readLine()); //sijoitetaan rivin
sisältö QStringListiin
    }

    delete file; // tuhotaan file olio
    return content; //palautetaan tiedoston sisältö kysy-
jälle
}
```

Kun tiedostoon kirjoitetaan, aukaistaan tiedosto kirjoitusoikeuksin. Tekstin siirtämiseen käytetään QTextStream-oliota samalla tavalla kuin luettiin edellisessä esimerkissä.

```
QFile file("out.txt");
file.open(QIODevice::WriteOnly | QIODevice::Text);
QTextStream out(&file);
out << "Tiedostoon kirjoitettava teksti.\n";
```

# 8 TOTEUTUS

## 8.1 Esitutkimus

Käyttäjätiedon keräämisessä pätee samat perusaskeleet, kuin muissakin ohjelmistokehityksen projekteissa. Kertaluontoisissa projekteissa yksinkertaisinta on seurata johdonmukaisesti vesiputousmallia, jonka ensiaskeleet ovat esitutkimus ja vaatimusten määrittely. Työn alussa pidimmekin työn ohjaajien kanssa palaverin, jossa kirjasin ylös asiakasyrityksen tarpeita. Keskustelimme, millaista tietoa on mahdollista kerätä ja miten se toteutettaisiin. Tässä on lista mitattavista tiedoista, joiden pohjalta suunnittelin tietokannan:

- paikka, jossa laitetta käytetään
- käyttäjän ikä
- käyttäjän kansallisuus
- tunniste, jolla sama käyttäjä voidaan tunnistaa eri paikoissa käyttävän palvelua
- Facebookiin lähetettyjen kuvien määrä
- otettujen kuvien määrä.

## 8.2 Laitteiden yksilöinti

Esitutkimuksessa ilmeni käyttäjätietojen tarpeettomuus, jos niitä ei voida erottaa eri laitteiden ja paikkojen välillä. Jokainen laite on todennäköisesti eri asiakkaalla ja käyttäjätieto on nimenomaan asiakkaalle tärkeätä. Päädyttiin identifioimaan laitteet uniikilla tunnisteella, jota yritys voi käyttää myös muiden laitteiden identifioimiseen. Ihmissilmälle pelkistä merkeistä koostuva tunniste on vaikea erottaa. Siksi tietokantaan lisättiin attribuutti, johon järjestelmänvalvoja voi kirjoittaa avaintietoa laitteesta, esimerkiksi paikkakunnan ja liikkeen nimen, missä laite sijaitsee. Näitä ei eroteltu omiksi attribuuteikseen, koska haluttiin säilyttää kuvauksen vapaus, eivätkä kaikki tiedot välttämättä ole ihan yksiselitteisiä. Tietokanta on siltä osin ristiriidassa ensimmäisen normaalimuodon kanssa.

Laitteen ID on yksilöllinen ja pysyvä, joten se on hyvä tallentaa paikkaan, joka ei ole käyttäjälle helppopääsyinen. Hyvänä paikkana ajattelin sijoittaa ID:n Linux-

ympäristömuuttujaksi. Sieltä se on helppo lukea, mutta sen muuttamiseen tarvitaan järjestelmänvalvojan oikeudet. Kuva lähetetään Facebookiin palvelimelta, jossa Facebook-sovellus sijaitsee. Kuvan nimenä käytetään laitteen ID:tä. Siten useammasta laitteesta on mahdollista lähettää kuvia yhtä aikaa, eivätkä ne sekoitu lähetysvaiheessa. Lisäksi kuvien poisto palvelimelta helpottuu, kun laitteen tunniste on sama kuin lähtevän kuvan nimi.

Mypose ohjelmassa ID:n lukemisesta vastaa staattinen funktio getPoseID(). Se on toteutettu samalla periaatteella kuin luvussa 6.3 oleva esimerkki.

```
QString Statistics::getPoseID()
{
    QString idString = "";
    QString prog = "/bin/bash";
    QStringList arguments;
    arguments << "-c" << "echo $ID";
    QProcess* process = new QProcess();
    process->start(prog , arguments);
    process->waitForFinished();
    idString = process->readAll();
    if(idString.length() > 0)
    {
        idString.chop(1);
    }
    delete process;
    if (idString.length() == 10)
    {
        return idString; //
    }
    else
    {
        return QString("unassigned");
    }
}
```

### 8.3 Käyttäjätiedon keräys Facebook-sovelluksella

Facebook-sovellusta kutsuttaessa lähetetään kutsun mukana kaksi tietoa, laitteen tunniste ja siirrettävään kuvaan liitettävä viesti. Tämä ei vielä riitä, sillä kun käyttäjä kirjautuu tunnuksillaan sisään, käy verkkoselain välissä useassa osoitteessa.  Tämä voidaan ratkaista luomalla evästeet molemmille muuttujille.

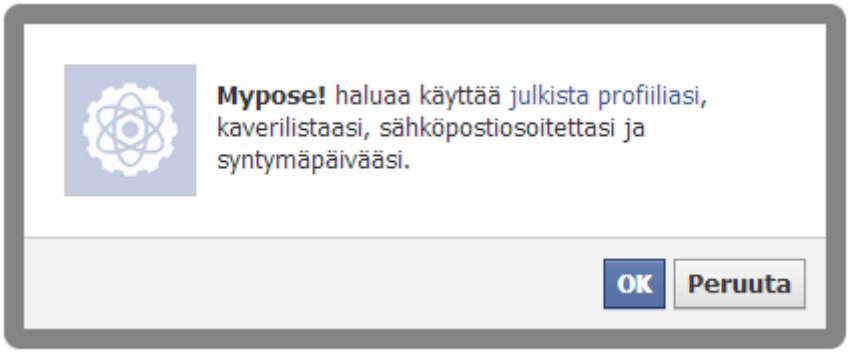
Evästeet poistuvat, kun selain tuhoutuu. Evästeet luodaan samoin, kuin luvussa 5.3 on esitetty.

```
if ($deviceID) {

  setcookie("identificationData","$deviceID");

  setcookie("messageData","$importMsg");

 }
```

Kun selain palaa takaisin sivulle, jolla sovellus on, lukee se evästeistä tunnisteen ja viestin.

Suurin osa kerättävistä tiedoista on suoraan saatavissa ilman erillistä lupaa, mutta syntymäpäivään pitää kysyä erillinen lupa käyttäjältä. Ohjelman toimivuuden kannalta on myös pyydettävä publish_stream-oikeus, jotta kuva voidaan siirtää käyttäjän kuvakansioon. Luvat pyydetään samalla, kun käyttäjä hyväksyy käyttöehtosopimuksen. Scope lisätään acces_tokenia pyydettäessä, jos tarvitaan ylimääräisiä lupia. Kuva 4 näyttää scopeen lisätyn publish_stream-oikeuden kysymisen.

```
if(empty($code)){

  $dialog_url = "http://www.facebook.com/dialog/oauth?"

   . "client_id=" .  $app_id

   . "&redirect_uri=" . urlencode($post_login_url)

   .  "&scope=publish_stream, user_birthday";

  echo("<script>top.location.href='" . $dialog_url . "'</script>");

 }
```




*KUVA 4. Käyttöehtojen kysyminen käyttäjältä*

Kun käyttäjä hyväksyy käyttöehtosopimuksen, voi hän käyttää palvelua ja ohjelma hakee hänestä käyttäjätietoja. Syntymäpäivä ei ole määrittelyn mukaista tietoa, joten se muutetaan calc_age()-metodilla ikävuosiksi.

```
if ($user)    {

$id = $user_profile['id'];

                $gender = $user_profile['gender'];

                $locale = $user_profile['locale'];

                $birthday = $user_profile['birthday'];

                $agge = calc_age ($birthday);

                }

function calc_age ($birth)

 {

   list($month,$day,$year) = explode("/",$birth);

   $year_diff  = date("Y") - $year;

   $month_diff = date("m") - $month;

   $day_diff   = date("d") - $day;
```

```
   if ($month_diff < 0) $year_diff--;

   elseif (($month_diff==0) && ($day_diff < 0)) $year_diff--;

   return $year_diff;

 }
```

### 8.4 Kuvamäärien laskeminen

Otettujen kuvien määrää seurataan sitä myötä, kun niitä otetaan, ja määrät tallennetaan kahden tunnin välein tietokantaan. Jos jostakin syystä laite ei ole ollut verkossa, kuvat tallennetaan jälkeenpäin sille päivälle, jolla ne on otettu kello 24:n kohdalle. Lisäksi, jos ohjelma kaatuu, hävikkiä numeroissa ei haluta syntyvän. Toteutukseen käytettiin tiedostoon kirjoitusta ja sieltä lukemista. Statistiikasta vastaava olio käy lukemassa tiedoston sisällön QStringListiin ja lähettää jonosta yksi päivitys kerrallaan tietokantaan lähettämättä jääneitä kuvamääriä. Jos lähetys onnistuu, ohjelma poistaa jonosta lähetetyn tiedon ja kirjoittaa jäljelle jääneen jonon tiedostoon. Sitten lähetetään seuraava tieto jonosta. Tätä jatkuu kunnes jono on tyhjä. Näin vältetään lähettämästä samaa tietoa kahteen kertaan. Jos lähetyksessä tapahtuu virhe, jätetään jono ennalleen ja yritetään seuraavan kerralla, kun kahden tunnin sykli tulee täyteen. Field-muuttuja määrää attribuutin, mihin kuvien määrä tallennetaan tietueessa.

```
mFILEstatistics = new QFile(this);
QString filename(QApplication::applicationDirPath() +
QDir::separator() + "statistics.txt");
mFILEstatistics->setFileName(filename);
mFILEstatistics->open(QIODevice::ReadWrite);
QTextStream in(mFILEstatistics);

while(!in.atEnd())
{
    mSLSTqueue.append(in.readLine());
}
sendData();


void Statistics::sendData()
```

```
{
    if(mSLSTqueue.size() < 1)
    {
        return;
    }
    QUrl url("");
    QUrl params;
    QString temp = mSLSTqueue.at(0);
    QStringList sndData = temp.split("$",
QString::SkipEmptyParts);
    mINTnextEvenHour = getEvenHour();
    if(sndData.at(1) !=
QDate::currentDate().toString("yyyy-MM-dd"))
    {
        params.addQueryItem("deviceID", getPoseID());
        params.addQueryItem("field", "24");
        params.addQueryItem("count", sndData.at(0));
        params.addQueryItem("date", sndData.at(1));
    }
    else
    {
        params.addQueryItem("deviceID", getPoseID());
        params.addQueryItem("field",
QString::number(mINTnextEvenHour));
        params.addQueryItem("count", sndData.at(0));
    }
    QByteArray bArray = params.encodedQuery();
    mNAMmanager->post(QNetworkRequest(url), bArray);
}
```

Tiedostossa kuvien määrä tallennetaan aina pareittain kuvanottopäivän kanssa. Tiedot erotetaan toisistaan dollarimerkillä. Aina kun kuva on otettu, kirjoitetaan uusin kuvien määrä. Kirjoittamista hoitaa WriteDataToStatisticsFile()-metodi. Sitä kutsutaan saman päivän kuvamäärän kanssa. Näin varmistetaan että ensimmäinen rivi on varmasti oikean arvoinen. Jos PictureTaken() kutsuu sitä parametrilla 0, tiedetään että kuva on ensimmäinen sille päivälle ja kirjoitetaan kokonaan uusi rivi. WriteDataToStatisticsFile() hoitaa myös tiedostoon kirjoittamisen silloin, kun tietokantaan on tallennettu tietoa, silloin metodia kutsutaan arvolla -1. Silloin kirjoitetaan ainoastaan jäljellä oleva jono tiedostoon.

```
void Statistics::PictureTaken()
{
    int count = 0;
    if(mSLSTqueue.size() > 0)
```

```
    {
        QStringList lastEntry =
mSLSTqueue.last().split("$", QString::SkipEmptyParts);
        if(QDate::currentDate().toString("yyyy-MM-dd") ==
lastEntry.at(1))
        {
            count = lastEntry.at(0).toInt() + 1;
        }
    }
    WriteDataToStatisticsFile(count);
}


void Statistics::WriteDataToStatisticsFile(int count)
{
    QTextStream out(mFILEstatistics);
    QString ref = "";
    out.seek(0);
    if(count > 0) // only happens when more pictures added
on current day
    {
        if(mSLSTqueue.size() > 1)
        {
            ref = mSLSTqueue.at(0);
            for(int i = 1; i <= mSLSTqueue.length()-2; i++)
            {
                ref = ref + "&" + mSLSTqueue.at(i);
            }
            ref.append("&" + QString::number(count) + "$" +
QDate::currentDate().toString("yyyy-MM-dd"));
        }
        else
        {
            ref = QString::number(count) + "$" +
QDate::currentDate().toString("yyyy-MM-dd");
        }
        mSLSTqueue.clear();
        mSLSTqueue = ref.split("&",
QString::SkipEmptyParts);
    }
    else if(count == 0)
    {
        mSLSTqueue.append("1$" +
QDate::currentDate().toString("yyyy-MM-dd"));
    }
    clearStatisticsFile();
    for(int i = 0; i < mSLSTqueue.size(); i++)
    {
        out << mSLSTqueue.at(i)  << "\r\n";
    }
}
```

Jotta kahden tunnin sykli saadaan oikeaan aikaan ja kuvien määrät vastaavat oikeita aikoja tietokannassa, lasketaan getFirstInterval()-metodilla aika, milloin pitää QTimer käynnistää. Metodi tarkastaa, mihin kahden tunnin kellojaksoon on sekunteja alle 7200 eli kahden tunnin sekuntien määrä. Sitten se palauttaa kyseisen määrän millisekunteina, josta vähennetään kahden minuutin varoaika. Varoaika estää tunnin vaihtumisesta koituvia ongelmia. Jos sekunteja on alle 120, palautetaan arvo 0.

```
int Statistics::getFirstInterval()
{
    for (int i=12; i > 0; i--)
    {
        int sec =
qAbs(QTime::currentTime().secsTo(QTime(i*2, 0, 0, 0)));
        if(sec > 120 && sec < 7200)
        {
            return (sec-120)*1000;
        }
        else if(sec <= 120)
        {
            return 0;
        }
    }
    return 0;
}
```

Jotta tiedetään, mihin attribuuttiin kuvien määrä lähetetään, lasketaan seuraava kaksituntinen. Se onnistuu metodilla getEvenHour(). Parillisuuden laskemiseksi käytetään tunnin jakojäännöstä apuna.

```
int Statistics::getEvenHour()
{
    int hour = QDateTime::currentDateTime().time().hour();
    if(hour%2 != 0)
    {
        return hour + 1;
    }
    else
    {
        return hour + 2;
    }
    return 24;
}
```

Sähköpostia lähettävä olio lähettää signaalin, joka kertoo kuinka moneen osoitteeseen sähköpostia on lähetetty. Se poimitaan statistiikkaluokassa ja lähetetään palvelimelle pyyntö päivittää tietokantaa.

```
void Statistics::mailsent(int pics)
{
    if(pics != 0)
    {
        QNetworkRequest req;
        req.setUrl(QUrl(""));
        QByteArray ba;
        QUrl params;
        params.addQueryItem( "de-
viceID",Statistics::getPoseID());
        params.addQueryItem( "picQuanti-
ty",QString::number(pics));
        ba = params.encodedQuery();
        mNAMmanager->post(req, ba);
    }
}
```

### 8.5 Tietojen tallennus tietokantaan

Qt:llä olisi mahdollista kirjoittaa tietoja suoraan tietokantaan, mutta päädyimme käyttämään palvelimella olevia PHP-kielisiä verkkosivuja datan siirtämiseen. Perusteluina tällaiseen ratkaisuun oli ajatus siitä, että toisella ohjelmointikielellä toteutettu ohjelma voisi myös tallentaa tietoja samoja verkkosivuja hyväksikäyttäen. Myös nykyisen ohjelman muuttuessa rajusti, on tietokantatoteutus valmiina.

PHP ottaa muuttujat vastaan POST()-metodia käyttäen. Sitten se suorittaa joukon tarkistuksia, jotta saatu data on varmasti oikean tyyppistä ja ettei kukaan yritä käyttää verkko-osoitetta väärin. Jos tiedot läpäisevät tarkistukset, luodaan yhteys tietokantaan. Verkkosivulle tulostetaan teksti ”premium_counter”, jotta ohjelma tunnistaa miten tulee käsitellä verkkovastausta.

**$ID = $_POST['deviceID'];**

**$COLUMN = $_POST['field'];**

```
$COUNT = $_POST['count'];

$PIC_DATE = $_POST['date'];

$DATE = date("Y-m-d");

if ($ID == "")

{

        $ID="unassigned";

}

if (strlen($PIC_DATE) == 0)

{

        $PIC_DATE = $DATE;

}

echo "premium_counter";


if(is_numeric($COUNT) && strlen($ID) == 10 && is_numeric($COLUMN))

{

        $connect = @mysql_connect (localhost, $username, $pass-
word);

}
```

Jos yhteys on luotu onnistuneesti, valitaan tietokanta ja selvitetään, onko kyseiseltä laitteelta tietuetta kyseiselle päivälle. Tästä riippuu kyselyn sisältö. Jos laitteella on samalle päivälle tietue, tarvitsee vain päivittää sitä. Muissa tapauksissa luodaan uusi tietue.

```
if($connect) {
```

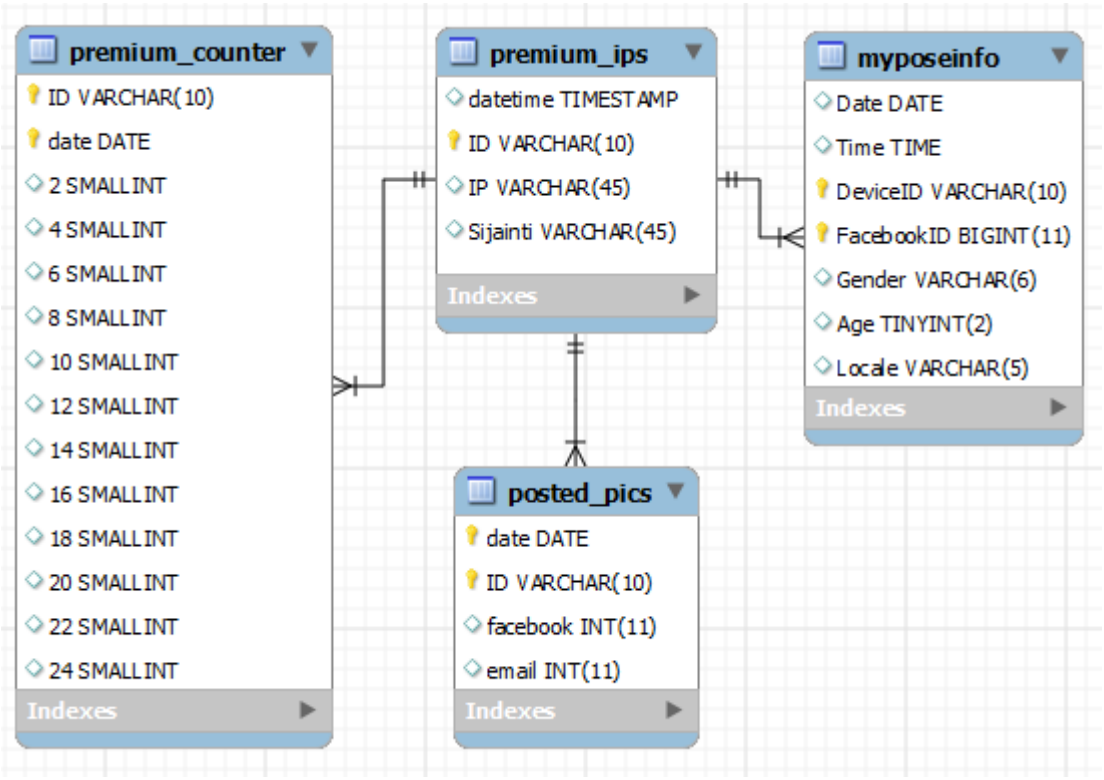
```
mysql_select_db( $database, $connect);


if($PIC_DATE == $DATE  || strlen($PIC_DATE) !=10)

{

$result = mysql_query("SELECT * FROM premium_counter WHERE ID='$ID' AND date='$DATE'");

$row = mysql_fetch_array($result);

if($row['ID'] == "")

{

$query = "INSERT INTO premium_counter (ID, date, `$COLUMN`) VALUES ('$ID', '$DATE', '$COUNT')";

}

elseif($row['ID'] == $ID && $row['date'] != $DATE)

{

$query = "INSERT INTO premium_counter (ID, date, `$COLUMN`) VALUES ('$ID', '$DATE', '$COUNT')";

}

else

{

$query = "UPDATE premium_counter SET `$COL-UMN`=`$COLUMN` + '$COUNT' WHERE ID='$ID' AND date='$DATE'";

}
```

```
}
else
{
$result = mysql_query("SELECT * FROM premium_counter WHERE ID='$ID' AND date='$PIC_DATE'");
$row = mysql_fetch_array($result);
if($row['ID'] == "")
{
$query = "INSERT INTO premium_counter (ID, date, `24`) VALUES ('$ID', '$PIC_DATE', '$COUNT')";
}
elseif($row['ID'] == $ID && $row['date'] != $PIC_DATE)
{
$query = "INSERT INTO premium_counter (ID, date, `24`) VALUES ('$ID', '$PIC_DATE', '$COUNT')";
}
else
{
$query = "UPDATE premium_counter SET `24`= `24` + '$COUNT' WHERE ID='$ID' AND date='$PIC_DATE'";
}
}
```

```
        mysql_query($query);

        mysql_close();
}
```

### 8.6 Tietokannan rakenne

Tietokanta on rakennettu neljästä taulusta, joista premium_ips toimii äititauluna kolmelle muulle taululle. ID on viiteavain jokaisessa taulussa. Viite-eheyssäännöiksi päätin päivityksen yhteydessä muutoksen vyöryttämisen (cascade) ja poiston yhteydessä eston (restrict). Premium_ips-taulun perusavain on ID, joka on myös osa perusavainta kaikille lapsitauluille. Premium_counter pitää ylhäällä päivän mittaan otettuja kuvia kahden tunnin jaksoissa. Sille luonnollinen valinta perusavaimeksi, on laitteen tunnisteesta (ID) ja päivämäärästä (date) muodostuva kokonaisuus. Normaalisäännöistä poiketen tässä taulussa on toistoa, mutta tässä tapauksessa se on perusteltua, koska on tarve saada samanlaista informaatiota tallennettua helposti luettavassa muodossa. Olisi hyvin epäkäytännöllistä luoda jokaisesta kahden tunnin jaksosta oma taulunsa. Myös posted_pics-taulun perusavain koostuu päivämäärästä ja laitteen tunnisteesta. facebookinfo-taulu ei ole riippuvainen päivämäärästä, vaan käyttäjästä (facebookID) ja laitteen tunnisteesta (DeviceID). Päivämäärä ja kellonaika on silti hyvä tallentaa, jotta tiedetään milloin laitetta on viimeksi käytetty ja onko esimerkiksi ikä edelleen validi tieto. Kuvassa 5 esitettävä ER-kaavio kertoo tarkemman tietokannan rakenteen.




*KUVA 5. Luodun tietokannan ER-kaavio. Piirretty MySQL Workbenchillä*

# 9 YHTEENVETO

Työn tarkoituksena oli tuottaa Mypose-ohjelmalle ohjelmistokokonaisuus, joka kerää asiakkaista tietoa kun he käyttävät laitetta. Tiedon tallentamisen tuli olla automaattista ja vastata asiakasyritysten tarpeita. Tallennusta varten oli suunniteltava tietokanta ja otettava selvää kuinka asiakastietoja voitaisiin kerätä. Lisäksi oli tutkittava millaista tietoa voi lain ja Facebookin käyttöehtojen mukaan kerätä ja mihin tarkoitukseen.

Kuvien lähetystä varten tehtiin sovellus käyttäen Facebookin PHP SDK:ta. Sovellus käytti PHP:n post()-metodia ja evästeitä viestien ja kuvien lähettämiseen. Toiminnan kannalta oli tärkeää lisätä scope-muuttujaan tietojen avaimet, joita ei oletuksena käyttäjältä saatu. Näitä oli syntymäaika ja julkaisuoikeudet. Facebookiin lähetetyistä kuvista saatiin irti käyttäjätietoa, kuten ikä, sukupuoli, käytetty kieli ja FacebookID. Ne tallennettiin tietokantaan käyttäen PHP:n MySQL-tukea. Laitteille luotiin tunnisteet, jotta ne ovat erotettavissa. Tunnisteella onnistuttiin välttämään ristiriitoja, joita tuli aiemmin usean laitteen käyttäessä samaan aikaan Facebook-ominaisuutta.

Kerättiin myös tietoja jotka koskivat yleisesti laitteen käyttöä, kuten montako kuvaa kullakin laitteella otetaan päivän mittaan. Tietoa tulee siis todella paljon, jokaiselle laitteelle päivittäin yksi tietue, jota päivitetään kerran kahdessa tunnissa. Työssä tutustuttiin Qt:n mahdollisuuksiin suorittaa käyttöjärjestelmäkomentoja ja toteutettiin erilaisia funktioita tiedon tallentamisen tueksi. Näistä mainittavimpia ovat tässä työssä esitetyt metodit getPoseID(), writeDataToStatisticsFile(), readFile(), getFirstInterval(), nextEvenHour() ja sendData(). Qt:n tärkeimmiksi työkaluiksi muodostuivat QTimer-, QFile-, QString-, QStringlist-, QProcess-, QFtp-, QAccessManager-, QNetworkReply- ja QNetworkRequest-luokat ja niiden metodit. Verkkotoiminnoista vastaavat jälkimmäiset neljä luokkaa.

Lisäksi toteutettiin yksinkertainen PHP-kielinen pohja, joka lukee tietokannasta laitteiden tietoja. Tiedot näytettiin taulukossa. Tämä toteutettiin ylimääräisenä opinnäytteeseen. Sittemmin tuo verkkosivu on jatkokehitetty täyttämään hakutarpeet tietokannasta.

Mahdollisia ongelmia on pyritty vähentämään sitä mukaan, kun niitä on ilmennyt. Esimerkiksi tunnisteen luominen poisti kuvien siirtymisen väärille henkilöille Facebookissa. Kuvan poistuminen on varmempaa, kun kuvan poistosta vastaavaa PHP:tä kuunnellaan QNetworkReplyllä. Jos kuva kuitenkin jäisi palvelimelle, ei kukaan ylimääräinen henkilö voi sitä saada, sillä yhteyttä Facebookiin ei luoda, mikäli kuva ei ole siirtynyt oikein QFtp:n välityksellä. QFtp on määritelty niin että se ylikirjoittaa olemassa olevan tiedoston. Jos jollakin laitteella ei ollut tallennettua tunnistetta, tuli tietokantaan useita rivejä. Se on ratkaistu tarkastamalla tallennettava data.

En ole täysin tyytyväinen sähköpostiin lähetettyjen kuvien laskemisesta. SMTP (simple mail transfer protocol) -palvelin ei lähetä virheilmoitusta, jos useaan sähköpostiin lähetetään viestejä ja yksi tai useampi lähetyksistä epäonnistuu. Jos yksikin onnistuu, saadaan palvelimelta ainoastaan viesti, että viestin lähetys onnistunut. Jouduttiin siis laskemaan kaikkiin samaan aikaan lähetettyihin sähköposteihin lähetyksen onnistuneeksi. Tämä voi vaikuttaa hieman todellisten välitettyjen viestien määrän poikkeavan lähetettyjen viestien määrästä. Tietokantaan tallennetaan lähetettyjen viestien määrä, koska todellisista määristä emme saa tietoa. Tietenkään ei lasketa lähetyksiä, joissa kaikki lähetykset epäonnistuvat. Suurin syy lähetysten epäonnistumisille ovat väärinkirjoitetut sähköpostiosoitteet.

Tavoitteet tavoitettiin onnistuneesti. Tietoa kertyy automaattisesti päivittäin, eikä se vaikuta käyttökokemukseen. Tieto on nimetöntä eikä loukkaa kenenkään yksityisyyttä. Kerättyä tietoa on jo käytössä Myposen omassa markkinoinnissa. Tunnistetta on hyödynnetty Myposen jatkokehityksessä, esimerkiksi laitteiden hallinnassa. Kaiken kaikkiaan työ on saatettu päätökseen ja sitä voi selkä suorassa katsella jälkeenpäin.

# LÄHTEET


1. Hyysalo, Sampsa 2009. Käyttäjä tuotekehityksessä. Helsinki: Taideteollinen korkeakoulu.
2. Royal Museums Greenwich. The history on naval logbooks. Saatavissa: http://www.rmg.co.uk/about/partnerships-and-initiatives/cliwoc/the-history-of-naval-logbooks. Hakupäivä 3.2.2013.
3. Kuniavsky, Mike 2003. Observing the user experience. Morgan Kaufmann Publishers.
4. Facebook Platform Policies. Saatavissa: http://developers.facebook.com/policy/. Hakupäivä 3.4.2013.
5. Codd, Edgar Frank 1970. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM vol 13, nro 6. S. 377–387.
6. Alaluukas, Pekka 2013. Relaatiotietokannat. Saatavissa: http://oamk.fi/~alaluuk/tietokannat/relaatiotietokannat.php. Hakupäivä 17.2.2013.
7. Tietokannan normalisoinnin perusteiden kuvaus. 2008. Saatavissa: http://support.microsoft.com/kb/283878/fi. Hakupäivä 24.2.2013.
8. SQL ORDER BY Keyword. Saatavissa: http://www.w3schools.com/sql/sql_orderby.asp. Hakupäivä 8.3.2013.
9. Ullman, Larry 2008. PHP 6 AND MySQL 5 for Dynamic Web Sites. Peachpit Press.
10. Rantala, Ari 2002. PHP. Docendo Finland Oy, Sanoma WSOY-konserni.
11. PHP Static variables. Saatavissa: http://www.tutorialspoint.com/php/php_static_variables.htm. Hakupäivä 9.3.2013.
12. PHP GET and POST Methods. Saatavissa: http://www.tutorialspoint.com/php/php_get_post.htm. Hakupäivä 9.3.2013.
13. Nixon, Robin 2009. Learning PHP, MySQL and JavaScript. O'Reilly Media, Inc.
14. mysqli::__construct mysqli_connect. Saatavissa: http://www.php.net/manual/en/mysqli.construct.php. Hakupäivä 18.3.2013.

15. mysql_connect. Saatavissa: http://www.php.net/manual/en/function.mysql-connect.php. Hakupäivä 18.3.2013.
16. mysql_select_db. Sattavissa: http://www.php.net/manual/en/function.mysql-select-db.php. Hakupäivä 18.3.2013.
17. mysql_query. Saatavissa: http://php.net/manual/en/function.mysql-query.php. Hakupäivä 19.8.2013.
18. Thelin, Jonathan 2007. Foundations of Qt Development. Apress.
19. QProcess Class Reference. Saatavissa: http://qt-project.org/doc/qt-4.8/qprocess.html#details. Hakupäivä 25.3.2013.

# LÄHTÖTIETOMUISTIO

Tekijä Jiri Lapinoja

Tilaaja Mypose Oy

Tilaajan yhdyshenkilö ja yhteystiedot

Lassi Anttonen 040 5166252

Työn nimi Statistikkapalikka

Työn kuvaus Työhön kuuluu selvittää millaisia statistiikkoja Pose-sovelluksella kerätään ja toteuttaa palikka, joka kirjoittaa ne tietokantaan luettavaksi sekä kerää ne.

Työn tavoitteet

Tavoiteaikataulu

Työn tarkoitus on olla valmiina 30.8.2012

Päiväys ja allekirjoitukset

23.2.2012

Jiri Lapinoja

Lassi Anttonen